μSystems Research Group

School of Electrical and Electronic Engineering

Towards asynchronous power management

Danil Sokolov, Andrey Mokhov, Alex Yakovlev, David Lloyd

Technical Report Series

NCL-EEE-MICRO-TR-2014-186

March 2014

Contact: danil.sokolov@ncl.ac.uk, andrey.mokhov@ncl.ac.uk, alex.yakovlev@ncl.ac.uk, david.lloyd@diasemi.com

NCL-EEE-MICRO-TR-2014-186

µSystems Research Group
School of Electrical and Electronic Engineering
Merz Court
Newcastle University
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

Towards asynchronous power management

Danil Sokolov, Andrey Mokhov, Alex Yakovlev, David Lloyd

March 2014

Abstract

Power management is an important part of modern microelectronics, however, the possibilities for its design automation are insufficiently studied and therefore the state-of-the-art synthesis methods produce suboptimal power control circuits. Currently the same design principles, which are based on synthesis of synchronous state machines, are used for both the data processing components and the power control circuits. While the synchronous operation is natural for data processing, it does not meet the low-latency and resilience requirements imposed by the power control logic. We believe that design of power control requires a fundamentally different approach based on clock-less design principles, which are characterised by robust operation in variable conditions and high responsiveness to the input stimuli. One of the main obstacles on this pathway is the difficulty of expressing the intended power control behaviour in a formal and unambiguous form which can be subsequently used for logic synthesis and verification of the obtained solution.

1 Introduction

The market of consumer gadgets is dominated by digital electronics that processes discrete data. However, a small portion of components remain analogue to operate on continuous values, such as the energy flows. As energy becomes the most valuable resource in modern electronics, the efficient implementation of such analogue components as power converters [1] is paramount for a wide range of applications, from extending the battery life of mobile gadgets to reducing the energy bill of large data centres. Responsiveness and robustness of power converters heavily depends on the implementation of their digital control circuitry – millions of control decisions need to be made each second and a single incorrect decision may cause a malfunction of the whole system or even permanently damage the circuit [2].

The practical design problem associated with power converters [3] are partially related to the state-of-the-art synthesis methods which produce suboptimal solutions. Currently the same design methods and CAD tools are used for building both the data processing components and the power control circuits. Historically these methods and tools are optimised for synchronous circuits whose bursts of activity are driven by the frequency of a global clock signal [4]. The clocked mode of operation is natural for the data processing, however, when applied to the power control it leads to either low responsiveness or power consumption overheads. On the one hand the clocking frequency must be extremely fast to capture all the tiny changes in the analogue power converter, which, on the other hand causes the waste of energy (useless switching of the global clock circuitry) when there are no changes to track.

The power control could significantly benefit from the use of asynchronous logic [5] which does not rely on the global clock signal and operates at the pace determined by the current situation. The key point here is

that such circuits are robust to variations in the operating conditions (e.g. voltage and temperature fluctuations) and are adaptable to the rate of changes in the controlled system [6]. This is already realised by the analogue engineers who are keen to use asynchronous circuits, but face the lack of design methods and tool support. Therefore they perform a very ad hoc design of power control circuits, never use formal specifications, and hence cannot prove correctness of their designs.

To bridge this gap one needs to develop a methodology and CAD tools for unambiguous representation of the design intents. Such a formal specification can be subsequently used for synthesis of the power control circuits in a correct-by-construction manner and for verifiable integration of the obtained control circuit into an analogue-digital system. The paper tackles these issues by proposing a methodology for design of the power control logic based on a specification language of labelled Petri nets [7] and its automated synthesis into an asynchronous circuit [8, 9]. On this pathway we try to maximally reuse the existing synthesis tools, and apply them to a novel domain of analogue electronics. A basic buck converter is used as a running example to demonstrate the whole design process.

The rest of the paper is organised as follows. Section 2 introduces our formalism for circuit specification. Section 3 overviews a basic power regulator and a way of specifying its control logic. This specification is used in Section 4 for synthesis and verification of the control circuitry. Section 5 outlines ideas for future work.

2 Petri nets and signal transition graphs

A well established modelling tool for capturing the causality and concurrency aspects of asynchronous circuits is the Signal Transition Graphs (STGs) [10] which is a special kind of Petri nets [11] whose transitions are associated with signal events.

Formally, a Petri net is defined as a tuple $PN = \langle P, T, F, M_0 \rangle$ comprising finite disjoint sets of *places* P and *transitions* T, *arcs* denoting the flow relation $F \subseteq (P \times T) \cup (T \times P)$ and *initial marking* M_0. There is an arc between $x \in P \cup T$ and $y \in P \cup T$ iff $(x,y) \in F$. The *preset* of a node $x \in P \cup T$ is defined as $\bullet x = \{y \mid (y,x) \in F\}$, and the *postset* as $x\bullet = \{y \mid (x,y) \in F\}$. The dynamic behaviour of a Petri net is defined as a *token game*, changing marking according to the enabling and firing rules. A *marking* is a mapping $M : P \to \mathbb{N}$ denoting the number of *tokens* in each place ($\mathbb{N} = \{0,1\}$ for *1-safe* Petri nets). A transition t is *enabled* iff $\forall p, p \in \bullet t \Rightarrow M(p) > 0$. The evolution of a Petri net is possible by *firing* the enabled transitions. *Firing* of a transition t results in a new marking M' such that $M'(p) = \begin{cases} M(p) - 1 & if\ p \in \bullet t \setminus t\bullet, \\ M(p) + 1 & if\ p \in t\bullet \setminus \bullet t, \\ M(p) & otherwise \end{cases}$ for all $p \in P$.

An STG is a 1-safe Petri net whose transitions are labelled by signal events, i.e. $STG = \langle P, T, F, M_0, \lambda, Z, v_0 \rangle$, where λ is a *labelling function*, Z is a set of *signals* and $v_0 \in \{0,1\}^{|Z|}$ is a *vector of initial signal values*. The labelling function $\lambda : T \to Z\pm$ maps transitions into *signal events* $Z\pm = Z \times \{+,-\}$. The signal events labelled $z+$ and $z-$ denote the transitions of signals $z \in Z$ from 0 to 1 (rising edge), or from 1 to 0 (falling edge), respectively. The labelling function does not have to be 1-to-1, i.e. transitions with the same label may occur several times in the net. In order to distinguish between transitions with the same label and refer to them from the text an index $i \in \mathbb{N}$ is attached to their labels as follows: $\lambda(t)/i$, where i differs for different transitions with the same label. STGs inherit the operational semantics of their underlying PNs, including the notions of transition enabling and firing.

Graphically, the places are represented as circles ○, transitions as boxes □, consuming and producing arcs

are shown by arrows $\longrightarrow$, and tokens are depicted by dots in the corresponding places $\odot$. For simplicity, the unmarked places with one transition in the preset and one transition in the postset are often hidden.

3 Buck converter

Delivery of energy and on-chip conversion of power levels is an important part of modern electronics. A basic power regulator comprises an analogue buck and its digital control logic, as shown in Figure 1. The control opens and closes the power regulating PMOS and NMOS transistors of the buck as a reaction to *under voltage* (UV), *over current* (OC) and *zero crossing* (ZC) conditions. These conditions are detected and signalled by a set of specialised sensors implemented as comparators of measured current and voltage levels against some reference values (I_max, V_0, V_ref).

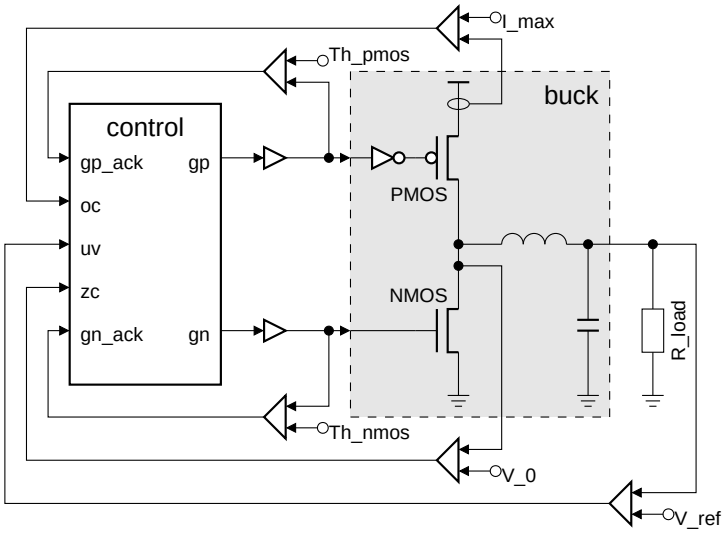


Figure 1: Power regulator

Note that the gp and gn signals are buffered to drive the very large power regulating transistors (occupy more than 50% of the buck area) and their effect on the buck can be significantly delayed. Therefore the controller is explicitly notified by the gp_ack and gn_ack signals when the power transistor threshold levels (Th_pmos and Th_nmos) are crossed on the buck side.

3.1 Specification of scenarios

The operation of a power regulator is usually specified in an intuitive, but rather informal way, e.g. by enumerating the possible sequences of detected conditions and describing the intended reaction to these events, as shown in Figure 2. The diagram reveals an alternation of the UV and OC conditions which are handled by opening and closing PMOS and NMOS transistors of the buck: in case of UV (resp., OC) the NMOS (resp., PMOS) transistor is switched off and the PMOS (resp., NMOS) is switched on. Detection of the ZC condition

after UV does not change this behaviour, however, if ZC is detected before UV then both the PMOS and NMOS transistors remain closed until the UV event. It is important to note that in order to avoid a short-circuit the PMOS and NMOS transistors of the buck must never be open at the same time. This is an important invariant that needs to be satisfied both by a specification and an implementation and can be verified within our approach, see Section 4. Note that while serving a descriptive purpose, this diagram does not provide a way of verifying that the specified scenarios are consistent with each other and do not introduce intrinsic conflicts.

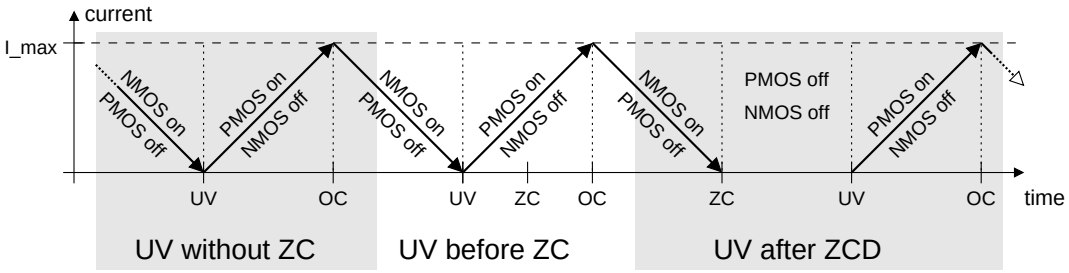


Figure 2: Informal specification of power control

A straightforward way of implementing this behaviour is by capturing its state machine in a standard RTL and synthesising it with a conventional EDA flow [4]. Taking into account the fact that the gp_ack and gn_ack inputs the just delayed versions of the gp and gn outputs, one can write the following synthesisable Verilog specification of the control logic:

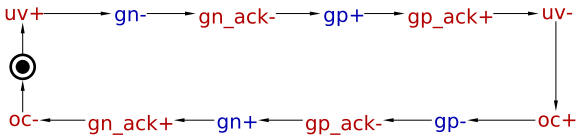
```
module control (clk, nrst, oc, uv, zc, gp_ack, gn_ack, gp, gn);
  input clk, nrst, uv, oc, zc, gp_ack, gn_ack;
  output reg gp, gn;
  always @(posedge clk or negedge nrst) begin
    if (nrst == 0) begin
      gp <= 0; gn <= 1;
    end else case ({gp_ack, gn_ack})
      2'b00: if (uv == 1) gp <= 1; else if (oc == 1) gn <= 1;
      2'b10: if (oc == 1) gp <= 0;
      2'b01: if (uv == 1 || zc == 1) gn <= 0;
    endcase
  end
endmodule
```

The RTL synthesis produces a relatively small control circuit comprising a pair of flip-flops for the gp and gn signals with a simple combinational logic in front. A major drawback of this approach is the dependency of the obtained circuit on the sampling frequency of the control inputs by an artificially introduced global clock signal. The higher is the clock frequency, the better is the responsiveness of the system. However, if the control inputs do not change for a long time, then the power is burned by the clock tree itself. Contrary, if the sampling frequency is low, then the response time of the control, which is measured in clock cycles, becomes unacceptably slow and may cause malfunctioning or even a permanent damage of the powered system.

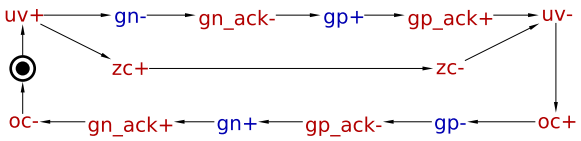
One can notice that for this type of control it is natural to remove the clock dependency altogether and build it in an asynchronous manner. Industry designers have already attempted this approach by assembling the control logic out of the library gates and validating its operation by exhaustive simulation. While this ad hoc approach works for relatively simple control circuits, it may become infeasible for complex controllers which need to handle multiple behavioural scenarios. A better alternative is to synthesise the asynchronous control circuit in a correct-by-construction way and subsequently verify its integration with the rest of the system. Additional invariants/properties can also be checked at this stage. Both the synthesis and verification rely on a formal specification of the desired control behaviour. Obtaining such a specification in a unambiguous form is the primary design automation challenge.

Consider deriving a formal specification of a basic buck control. According to the diagram of Figure 2 there are three distinctive scenarios to capture: (i) UV happens without ZC, (ii) UV is followed by ZC and (iii) UV happens after ZC. Let us capture one of the scenarios, e.g. when UV happens without ZC, in an STG form.

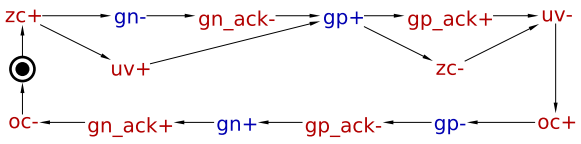
Initially the NMOS transistor is open (on state) and the PMOS transistor is closed (off state) which should lead to the UV condition. When UV is detected (uv+), the NMOS transistor needs to get closed (gn-). When the closing of NMOS is confirmed (gn_ack-) the PMOS transistor can be open to charge the buck (gp+ indicated by gp_ack+). Eventually the buck will saturate leading to OC (oc+) at which stage the PMOS needs to be closed (gp-). After the closure of the PMOS transistor is confirmed (gp_ack-) the NMOS transistor gets open (gn+ indicated by gn_ack+), leading to the release of OC (oc-) and brining us to the initial state. The resultant STG listing the sequence of signal events for this scenario is shown in Figure 3a. The scenarios for UV occurring before and after ZC are formalised in similar ways by the STGs in Figures 3b and 3c respectively.



(a) UV without ZC



(b) UV before ZC

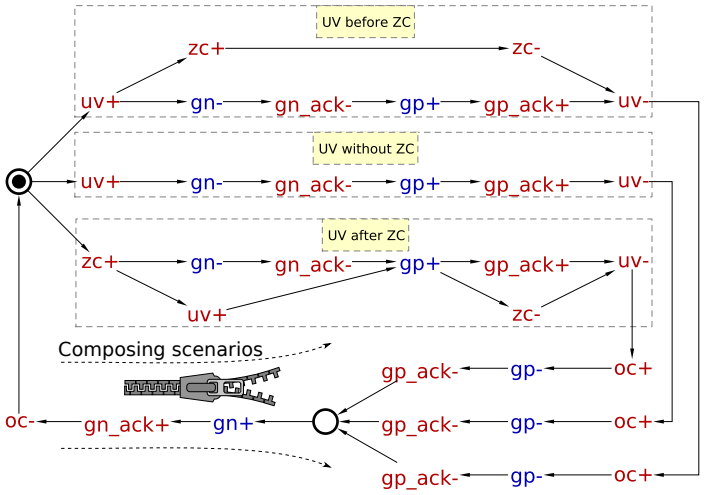


(c) UV after ZC

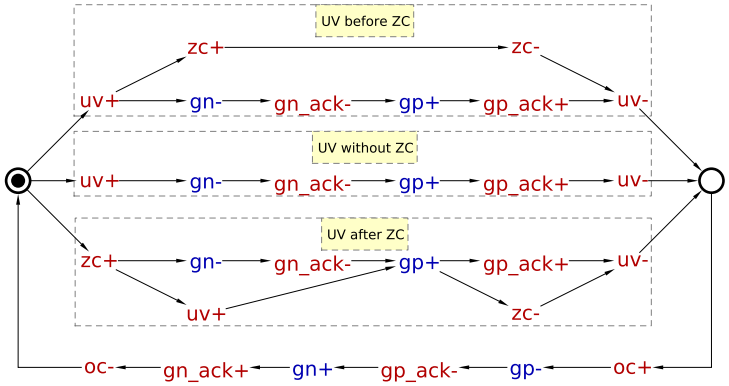
Figure 3: STG models for buck control scenarios

3.2 Composition

Figure 3 shows three behavioural scenarios of the buck operation. We can synthesise a circuit implementing a particular scenario from the corresponding STG. However, in order to produce an implementation capable of handling all of the scenarios, we need to compose them into a single specification. This can be a challenging problem [12] and we will not discuss the general solution in this paper due to the lack of space, but fortunately the specific case at hand can be solved without much difficulty.



(a) Composing scenarios



(b) Complete specification

Figure 4: Composition of scenarios into STG specification

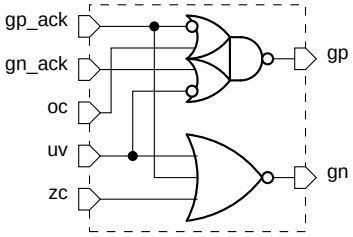
One can see that all three STGs have ‘compatible’ initial states, that is all common input and output signals are set to the same values initially. Note the following subtle point. Signal zc does not appear in the second scenario and we have to assume that its value is set to constant 0 as otherwise the whole multi-scenario specification becomes inconsistent. With this assumption at hand, we can merge the initially marked place in the three STGs, because it corresponds to the same global state in all three scenarios.

Once the initially marked places are merged, one can notice that we can also merge three transitions oc- leading to it because the preceding states are also compatible. This process continues with signal event gn_ack+, and so on, ‘zipping’ the common paths of the STGs together, as shown in Figure 4a. Finally, when we reach

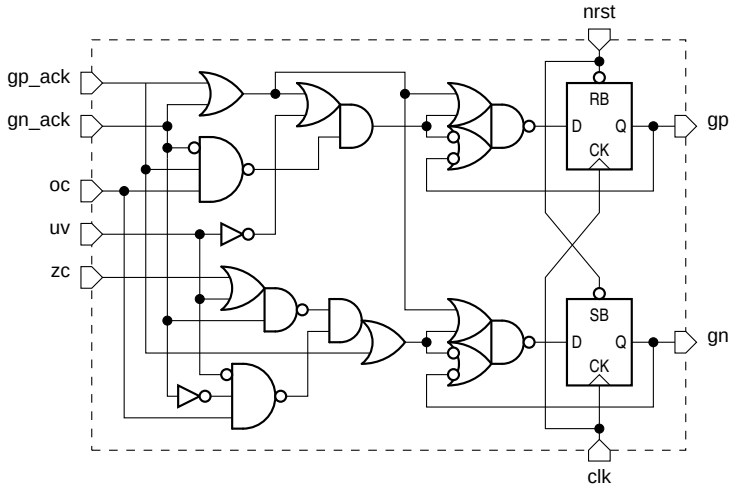
the point where the paths diverge we stop; see the resulting STG in Figure 4b. Note that in principle we could have also merged transitions uv-, but that would require adding dummy transitions to synchronise events zc- and gp_ack+ in the upper and lower branches of the resulting STG.

4 Synthesis and verification

An asynchronous complex gate solution obtained by Petrify [13] and optimised for negative gates by De Morgan's laws is shown in Figure 5a. Its complexity is much lower than the synchronous circuit obtained form RTL specification: only a couple of complex gates compared to a pair of flip-flops and half a dozen of gates in case of synchronous design synthesised from the RTL specification, as shown in Figure 5b. Moreover, the input-output latency of the asynchronous circuit is bounded by the delay of a single gate, while in synchronous design it is determined by the clock period – the reaction to the current state of the inputs is observed only in the next clock cycle.



(a) Complex gate asynchronous implementation



(b) Synchronous implementation

Figure 5: Buck control circuit

The Petrify solution is speed-independent [8], i.e. insensitive to delays of individual gates, which makes it robust to delay variations caused by temperature and voltage fluctuations. This property can be formally verified following the technique presented in [14]. A circuit is considered speed-independent under given environment,

if it is hazard-free and conforms to the environment (i.e. produces only those changes of output signals that do not conflict with the environment STG).

The circuit verification is done by reachability analysis of the composed system specification: the circuit implementation and expected behaviour of the environment, both expressed with STGs. At this stage other invariants and global properties can be also checked, e.g. that no state is reachable in the composed STG model where both PMOS and NMOS transistors are open.

If the library of available gates does not contain the required complex gates, then the circuit can be decomposed into simpler gates. Note that technology mapping of speed-independent circuits is a computationally hard problem as the decomposed circuit must remain hazard-free and insensitive to the delays of individual gates [15].

The complex gate solution for this relatively simple buck control is purely combinational and does not have any memory, therefore its off-line testing is straightforward. In general, to comply with testability requirements it is advantageous to synthesise the control logic using so-called generalised C-elements, that can be implemented in a testable way and integrated into a scan chain for conventional off-line testing [16].

5 Conclusions

Our research is concerned with developing and applying asynchronous design methods that were traditionally focused on digital systems, to the analogue world. Currently the major challenge is obtaining a formal specification of the desired control behaviour for automated implementation with existing logic synthesis tools. When this goal is achieved, the existing synthesis methods will be extended to make use of timing assumptions specific for the analogue world and translate them into the timing constraints. Verification of the circuit compliance with these constraints is a subject for future work.

References

[1] A. Pressman, K. Billings, T. Morey: *"Switching power supply design"*, 3rd edition, McGraw-Hill, 2009.

[2] J. Audy: *"Navigating the path to a successful IC switching regulator design"*, Tutorial at IEEE International Solid-State Circuits Conference (ISSCC), 2008.

[3] T. Towers: *"Practical design problems in transistor DC/DC converters and DC/AC inverters"*, Proc. IEE, vol. 106(18), pp. 1373–1383 1959.

[4] L. Scheffer, L. Lavagno, G. Martin: *"EDA for IC system design, verification, and testing (Electronic design automation for integrated circuits handbook)"*, CRC Press, 2006.

[5] S. Unger: *"Asynchronous Sequential Switching Circuit"*, Wiley-Interscience, 1969.

[6] J. Sparsø, S. Furber: *"Principles of asynchronous circuit design"*, Kluwer Academic Publishers, 2001.

[7] A. Yakovlev, L. Lavagno, A. Sangiovanni-Vincentelli: *"A unified signal transition graph model for asynchronous control circuit synthesis"*, Formal Methods in System Design, vol. 9(3), pp. 139–188, 1996.

[8] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev: *"Logic synthesis of asynchronous controllers and interfaces"*, Springer Series in Advanced Microelectronics, vol. 8, 2002.

[9] V. Khomenko, M. Koutny, A. Yakovlev: *"Logic synthesis for asynchronous circuits based on STG unfoldings and incremental SAT"*, Fundamenta Informaticae, vol. 70(1-2), pp. 49–73, 2006.

[10] L. Rosenblum, A. Yakovlev: *"Signal graphs: from self-timed to timed ones"*, Proc. International Workshop on Timed Petri Nets, pp. 199–206, 1985.

[11] C. Petri: *"Kommunikation mit automaten (Communicating with automata)"*, University of Bonn, PhD Thesis, 1962.

[12] A. Mokhov, A. Yakovlev: "Conditional partial order graphs: model, synthesis and application", IEEE Transactions on Computers, vol. 59(11), pp. 1480–1493, 2010.

[13] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev: *"Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers"*, IEICE Transactions on Information and Systems, vol. E80-D(3), pp. 315–325, 1997.

[14] I. Poliakov, A. Mokhov, A. Rafiev, D. Sokolov, A. Yakovlev: *"Automated verification of asynchronous circuits using circuit Petri nets"*, Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), pp. 161–170, 2008.

[15] V. Khomenko: *"Logic decomposition of asynchronous circuits using STG unfoldings"*, Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), pp. 3–12, 2011.

[16] D. Lloyd, R. Illman: *"Scan insertion and ATPG for C-gate based asynchronous designs"*, Synopsys User Group (SNUG), 2014 (to appear).